Information and Software Technology 99 (2018) 41–57

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

We're doing it live: A multi-method empirical study on continuous experimentation

Gerald Schermann*[,a], Jürgen Cito[a], Philipp Leitner[a], Uwe Zdun[b], Harald C. Gall[a]

[a] Department of Informatics, University of Zurich, Switzerland
[b] University of Vienna, Austria

ARTICLE INFO

Keywords:
Release engineering
Continuous deployment
Continuous experimentation
Empirical study

ABSTRACT

Context: Continuous experimentation guides development activities based on data collected on a subset of online users on a new experimental version of the software. It includes practices such as canary releases, gradual rollouts, dark launches, or A/B testing.
Objective: Unfortunately, our knowledge of continuous experimentation is currently primarily based on well-known and outspoken industrial leaders. To assess the actual state of practice in continuous experimentation, we conducted a mixed-method empirical study.
Method: In our empirical study consisting of four steps, we interviewed 31 developers or release engineers, and performed a survey that attracted 187 complete responses. We analyzed the resulting data using statistical analysis and open coding.
Results: Our results lead to several conclusions: (1) from a software architecture perspective, continuous experimentation is especially enabled by architectures that foster independently deployable services, such as microservices-based architectures; (2) from a developer perspective, experiments require extensive monitoring and analytics to discover runtime problems, consequently leading to developer on call policies and influencing the role and skill sets required by developers; and (3) from a process perspective, many organizations conduct experiments based on intuition rather than clear guidelines and robust statistics.
Conclusion: Our findings show that more principled and structured approaches for release decision making are needed, striving for highly automated, systematic, and data- and hypothesis-driven deployment and experimentation.

1. Introduction

Many software developing organizations are looking into ways to further speed up their release processes and to get their products to their customers faster [1]. One instance of this is the current industry trend to "move fast and break things", as made famous by Facebook [2] and in the meantime adopted by a number of other industry leaders [3]. Another example is continuous delivery and deployment (CD) [4]. *Continuous delivery* is a software development practice where software is built in such a way that it can be released to production at any time, supported by a high degree of automation [5]. *Continuous deployment* goes one step further; software is released to production as soon as it is ready, i.e., passing all quality gates along the deployment pipeline. These practices pave the way for controlled continuous experimentation (e.g., A/B testing [6], canary releases [4]), which are a means to guide development activities based on data collected on a subset of online users on a new *experimental* version of the software. Unfortunately, our knowledge of continuous experimentation practices is currently primarily based on well-known and outspoken industrial leaders [6,7]. This is a cause for concern for two reasons. Firstly, it raises the question to what extent our view of these practices is coined by the peculiarities and needs of a few innovation leaders, such as Microsoft, Facebook, or Google. Secondly, it is difficult to establish what the broader open research issues in the field are.

Hence, we conducted a mixed-method empirical study, in which we interviewed 31 software developers and release engineers from 27 companies. To get the perspective of a broader set of organizations, we specifically focused on a mix of different team and company sizes and domains. However, as continuous experimentation is especially amenable for Web-based applications, we primarily selected developers or release engineers from companies developing Web-based applications for our interviews. We combined the gathered qualitative interview data with an online survey, which attracted a total of 187 complete responses. The design of the study was guided by the following research

* Corresponding author.
E-mail addresses: schermann@ifi.uzh.ch (G. Schermann), cito@ifi.uzh.ch (J. Cito), leitner@ifi.uzh.ch (P. Leitner), uwe.zdun@univie.ac.at (U. Zdun), gall@ifi.uzh.ch (H.C. Gall).

https://doi.org/10.1016/j.infsof.2018.02.010
Received 28 March 2017; Received in revised form 21 February 2018; Accepted 27 February 2018
Available online 10 March 2018
0950-5849/

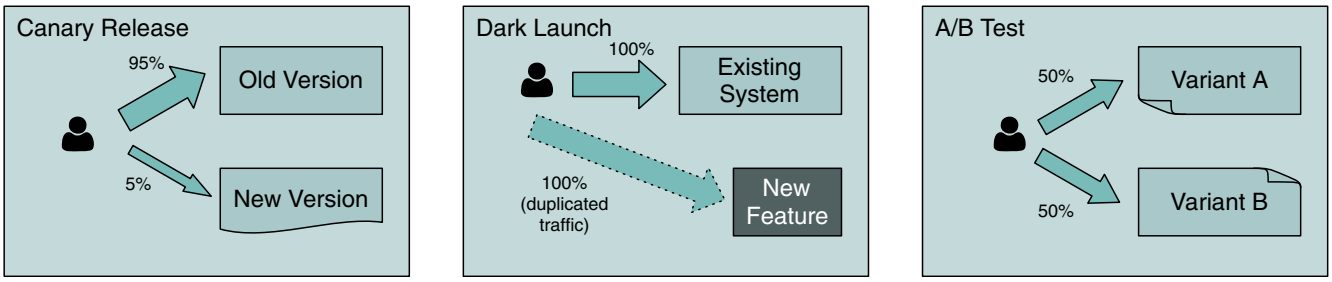

Fig. 1. Overview of canary releases, dark launches, and A/B testing.

questions.

***RQ1** : What principles and practices enable and hinder organizations to leverage continuous experimentation?*

We identified the preconditions for setting up and conducting continuous experiments. Continuous experimentation is facilitated through a high degree of deployment automation and the adoption of an architecture that enables independently deployable services (e.g., microservices-based architectures [8]). Important implementation techniques include *feature toggles* [9] and *runtime traffic routing* [10]. Experimenting on live systems requires more insight into operational characteristics of these systems. This requires extensive monitoring and safety mechanisms at runtime. Developer on call policies are used as risk mitigation practices in an experimentation context. Experiment data collection and interpretation is essential. However, not all teams are staffed with experts in all relevant fields, we have seen that these teams can request support from internal consulting teams (e.g., data scientists, DevOps engineers, or performance engineers).

***RQ2** : What are the different flavors of continuous experimentation and how do they differ?*

Having insights into the enablers and hindrances of experimentation, we then investigated how companies make use of experimentation. Organizations use different flavors of continuous experimentation for different reasons. *Business-driven experiments* are used to evaluate new functionality from a business perspective, first and foremost using A/B testing [6]. *Regression-driven experiments* are used to evaluate non-functional aspects of a change in a production environment, i.e., validate that a change does not introduce an end user perceivable regression. In our study, we have observed differences in these two flavors concerning their main goals, evaluation metrics, how their data is interpreted, and who bears the responsibility for different experiments. We have also seen commonalities in how experiments are technically implemented and what their main obstacles of adoption are.

Based on the outcomes of our study, we propose a number of promising directions for future research. Given the importance of architecture for experimentation, we argue that further research is required on architectural styles that enable continuous experimentation. Further, we conclude that practitioners are in need of more principled approaches to release decision making (e.g., which features to conduct experiments on, or which metrics to evaluate).

The rest of this paper is structured as follows. In Section 2, we introduce common continuous experimentation practices. Related previous work is covered in Section 3. Section 4 gives more detail on our chosen research methodology, as well as on the demographics of our study participants and survey respondents. The main results of our research are summarized in Sections 5 and 6, while more details on the main implications and derived future research directions are given in Section 7. Finally, we conclude the paper in Section 8.

2. Background

Adopting CD, thus increasing release velocity, has been claimed to allow companies to take advantage of early customer feedback and faster time-to-market [1]. However, moving fast increases the risk of rolling out defective versions. While sophisticated test suits are often successful in catching functional problems in internal test environments, performance regressions are more likely to remain undetected, hitting surface only under production workloads [11]. Techniques such as user acceptance testing help companies estimate how users appreciate new functionality. However, the scope of those tests is limited and allows no reasoning about the demand of larger populations. To mitigate these risks, companies have started to adopt various continuous experimentation practices, most importantly canary releases, gradual rollouts, dark launches, and A/B testing. We provide a brief overview of these experimentation practices in Section 2.1, followed by an introduction to two common techniques how these practices can be implemented in Section 2.2.

2.1. Experimentation practices

Fig. 1 illustrates the practices of canary releases, dark launches, and A/B testing.

Canary releases. Canary releases [4] are a practice of releasing a new version or feature to a subset of customers only (e.g., randomly selecting 5% of all customers in a geographic region), while the remaining customers continue using the stable, previous version of the application. This type of testing new functionality in production limits the scope of problems if things go wrong with the new version.

Dark launches. Dark, or shadow, launching [2,12] is a practice to mitigate performance or reliability issues of new or redesigned functionality when facing production-scale traffic. New functionality is deployed to production environments without being enabled or visible for any users. However, in the backend, “silent” queries generated based on production traffic are forwarded to the “shadow” version. This provides insights into how the feature would be behaving in production, without actually impacting users.

Gradual rollouts. Gradual rollouts [4] are often combined with other continuous experimentation practices, such as canary releases or dark launches. The number of users assigned to the newest version is gradually increased (e.g., increase traffic routed to the new version in 5% steps) until the previous version is completely replaced or a predefined threshold is reached.

A/B testing. A/B testing [6] comprises running two or more variants of an application in parallel, which only differ in an isolated implementation detail. The goal is to statistically evaluate, usually based on business metrics (e.g., conversion rate), which of those versions performed better, or whether there was a statistically significant difference at all.

2.2. Implementation techniques

The two common implementation techniques for conducting experiments are feature toggles and runtime traffic routing.

Feature toggles. Feature toggles [9] are a code-level experimentation technique. In their simplest form, they are conditional statements in the source code deciding about which code block to execute next (e.g., whether a certain feature is enabled for a specific